Big-O Notation Study Guide

Introduction to Big-O Notation

Big-O notation is used to describe the efficiency of an algorithm. It quantifies the worst-case scenario in terms of time complexity or space complexity as the input size grows. We analyze how the execution time (or memory usage) scales with increasing input size (n).

Common Big-O Complexities

Big-O	Category	Example
O(1)	Constant Time	Accessing an array element by index
O(log n)	Logarithmic Time	Binary search
O(n)	Linear Time	Looping through an array
O(n log n)	Log-Linear Time	Merge Sort, Quick Sort (average case)
O(n^2)	Quadratic Time	Nested loops (Bubble Sort, Insertion Sort)
O(2^n)	Exponential Time	Recursive Fibonacci calculation
O(n!)	Factorial Time	Brute-force permutations of a set

Understanding Big-O Computation

To determine the complexity of an algorithm, consider:

1. How many loops are present?
2. How does data growth affect operations?
3. What is the worst-case scenario?

Example: Consider a nested loop traversing a 2D array (n x n). Each loop runs 'n' times, resulting in O(n^2) complexity.

C++ Examples

O(1) - Constant Time

```
int getFirstElement(int arr[]) {
    return arr[0]; // Always constant time O(1)
}
```

O(n) - Linear Time

```
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << endl; // Runs 'n' times -> O(n)
    }
}
```

O(n^2) - Quadratic Time

```
void printPairs(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << arr[i] << ", " << arr[j] << endl; // Nested loop O(n^2)
        }
    }
}
```

Practice Problems

1. What is the Big-O complexity of finding an element in a sorted array using binary search?

2. Determine the complexity of a function with three nested loops iterating up to 'n'.

3. What is the time complexity of inserting an element into a sorted linked list?

4. Analyze the complexity of the following function:

```
void example(int n) {
    for (int i = 1; i < n; i *= 2) { cout << i; }
}
```

5. What is the best-case and worst-case time complexity of Quick Sort?

Answer Key

1. O(log n) - Binary search halves the input size each step.

2. O(n^3) - Three nested loops result in cubic complexity.

3. O(n) - Insertion into a sorted linked list requires traversal.

4. O(log n) - The loop doubles 'i' each iteration, leading to logarithmic growth.

5. Best-case: O(n log n) (pivot divides array evenly); Worst-case: O(n^2) (pivot always smallest/largest).